Journal of Heuristics, 8: 83–107, 2002

Constraint Propagation in Graph Coloring

MASSIMILIANO CARAMIA*
Dipartimento di Informatica, Sistemi e Produzione, University of Rome "Tor Vergata", Via di Tor Vergata, 110-00133 Rome, Italy
email: caramia@disp.uniroma2.it

PAOLO DELL'OLMO
Dipartimento di Statistica, Probabilità e Statistiche Applicate, University of Rome "La Sapienza", Piazzale Aldo Moro, 5-00185 Rome, Italy
email: dellolmo@pow2.sta.uniroma1.it

Abstract

In this paper we propose a method for integrating constraint propagation algorithms into an optimization procedure for vertex coloring with the goal of finding improved lower bounds. The key point we address is how to get instances of Constraint Satisfaction Problems (CSPs) from a graph coloring problem in order to give rise to new lower bounds outperforming the maximum clique bound. More precisely, the algorithms presented have the common goal of finding CSPs in the graph for which infeasibility can be proven. This is achieved by means of constraint propagation techniques which allow the algorithms to eliminate inconsistencies in the CSPs by updating domains dynamically and rendering such infeasibilities explicit. At the end of this process we use the largest CSP for which it has not been possible to prove infeasibility as an input for an algorithm which enlarges such CSP to get a feasible coloring. We experimented with a set of middle-high density graphs with quite a large difference between the maximum clique and the chromatic number.

Key Words: binary constraints, clique, edge consistency, heuristic algorithms, lower bound, vertex coloring

1. Introduction

It is well known that many combinatorial optimization problems have been modeled as Constraint Satisfaction Problems (CSPs): school timetabling, scheduling, line-drawing, graph labeling, sketch-mapping, interpretation and circuit design can be expressed in a natural way as CSPs (e.g., see Baptiste, Le Pape, and Nuijten, 1995; Dechter and Pearl, 1988). In this paper the vertex coloring problem is modeled as a succession of CSPs to give a further example of the integration of Operations Research and Artificial Intelligence techniques.

Let G be an undirected and connected graph, with vertex (node) set V and edge set E. Denote with $G_{V'}$ the subgraph induced by the node set $V' \subseteq V$ and with $E(G_{V'})$ its edge set. A *proper coloring* is a function $f : V \to \{1, 2, \ldots, |V|\}$ such that $f(i) \neq f(j)$ for each $(i, j) \in E$, i.e., no two adjacent nodes are assigned the same color. The smallest number of colors for which a proper coloring of G exists is called *chromatic number* and is denoted

*Author to whom all correspondence should be addressed.

by $\chi(G)$. The problem of finding the chromatic number of a graph is $\mathcal{NP}$-hard (Garey and Johnson, 1976), and has stimulated a large number of papers, dealing with both exact algorithms (e.g., see Caramia and Dell'Olmo, 2001a; Mehrotra and Trick, 1996; Sager and Lin, 1993; Sewell, 1996), and heuristic algorithms (e.g., see Caramia and Dell'Olmo, 1999; Caramia and Dell'Olmo, and Italiano, 1999; Fleurent and Ferland, 1995, 1996; Johnson et al., 1991).

Define a *clique* as a subset $W \subseteq V$ of nodes that induces a complete subgraph in G: a clique is *maximum* if there does not exist a clique of larger cardinality in G and its size is denoted by $\omega(G)$. $\omega(G)$ is the most common lower bound on $\chi(G)$ used: in fact, to the best of our knowledge, there does not exist a combinatorial algorithm with the specific goal of finding a lower bound for coloring. However, using $\omega(G)$ as lower bound on $\chi(G)$ leaves room for some drawbacks. In particular, since the gap $\chi(G) - \omega(G)$ can be arbitrarily large (Tutte, 1954), such a lower bound can be far from the optimum value; moreover, also the problem of finding a maximum clique in a graph is $\mathcal{NP}$-hard (Garey and Johnson, 1979).

We believe that the first disadvantage has a deeper impact than the second, as coloring gets more difficult as the upper bound-lower bound gap embedded in an enumerative framework becomes higher, while for what appears in the literature (e.g., see Babel, 1991; Balas and Xue, 1996), instances for which the maximum clique problem can be solved exactly are substantially larger than those for which an exact coloring can be achieved.

Motivated by these considerations we have exploited a clique (not necessarily maximum) as the starting point of an iterative and more general process based on constrained propagation to achieve improved lower bounds. For the sake of completeness, we recall basic notations and definitions of the CSP.

A CSP involves a set of n variables $X_1, X_2, \ldots, X_n$ and a set of n domains, $D_1, D_2, \ldots, D_n$ where each D_i defines the set of values that the variable X_i may assume. A solution of a CSP is an assignment of a value to each variable drawn in its domain which satisfies a given set of constraints. A binary CSP is one in which all constraints involve only pairs of variables. A binary constraint between variables X_i, X_j is a subset of the Cartesian product $D_i \times D_j$. A binary CSP can be associated with a constraint graph in which nodes represent variables and edges connect pairs of constrained variables.

Recalling that the coloring problem aims to assign the minimum number of colors to the nodes of a graph such that no monochromatic edge is allowed, there exists a one to one mapping between a constraint graph and a graph to be colored. In other words, graph coloring can be seen as a binary CSP where $X_1, \ldots, X_n$ are the variables that represent, respectively, the colors to be assigned to the n nodes in V, and in the constraint graph there is an edge between X_i, X_j, whenever the corresponding vertices $i, j \in V$ are adjacent in G. Thus, the binary constraints are of the form $\{(x, y), x \in D_i, y \in D_j : x \neq y\}$. For the sake of simplicity in the sequel we will call variable X_i as i.

The known attempts to model graph coloring as a CSP are devoted to finding feasible solutions to the former problem, and from what appears in the literature, given a graph coloring instance, the domain definition seems to be the most critical task. We believe this is true since constraints are directly defined by the edge set. Indeed, that is possibly the reason why in Cooper (1997) and Dechter and Pearl (1988) both the problems are mentioned but

not effectively related. The idea discussed in Cooper (1997) and Dechter and Pearl (1988) is to assign domains to variables in a very natural way: guess a value, say c, for which one supposes a feasible coloring exists and assign to each variable X_i, corresponding to the node $i \in V$, the same domain $D_i = \{1, \ldots, c\}$. Clearly, if c is much greater than the chromatic number one can expect to find a solution easily (i.e., an admissible coloring), which however, might be of very poor quality. Moreover, in this scenario (each node is assigned the same domain), reduction techniques, such as *neighborhood substitution* (Cooper, 1997), are not effective as their efficiency can be proved only if domains are assigned to nodes with a random choice on the set $\{1, \ldots, c\}$. In other words, it seems that if one decides to assign the same domain to all the nodes, one can no longer exploit the graph structure or particular constraint properties. We note also that no specific ordering for the assignment of variables is given.

The above mentioned difficulties encountered in the designing of the domains have led us to *avoid* assignment of the same domain $D_i = \{1, \ldots, c\}$ to all nodes of the graph. Instead, we start from a properly chosen subset of nodes giving rise to a small CSP to be handled quite easily and which might still be representative of the whole problem. In particular, we activate this process by starting from a maximal clique W in G and present a number of progressively more sophisticated algorithms capable of giving rise to infeasibilities, when they exist, through an iterative constraint propagation. Moreover, we define an extension of the proposed approach when used in the direction of feasibility, presenting a further algorithm based on constraint propagation and relying on known conditions for the existence of backtrack free solutions in networks of binary constraints (Mackworth, 1976).

Our analysis concentrates on graphs whose chromatic numbers are strictly greater than the size of their maximum cliques. The reason is twofold: first, we believe they represent hard to color instances; second, if the chromatic number equals the maximum clique size one can use existing maximum clique algorithms to get the best lower bound achievable. Experimental results show that the starting lower bound offered by the cardinality of a maximal clique can be increased, improving with limited computational effort on $\omega(G)$.

The outline of the paper is the following. Section 2 describes constraint propagation algorithms devoted to the lower bound computation, while Section 3 gives a description of the algorithm to compute feasible CSP solutions (upper bounds). Section 4 contains computational results and some examples of the algorithm execution. Finally, Section 5 completes the paper with some conclusive remarks.

2. Checking for infeasibility: Lower bound calculation

The algorithms presented in this section have the common goal of finding CSPs in the graph for which it is possible to prove infeasibility. This is achieved by means of constraint propagation techniques which allow the algorithms to eliminate inconsistencies in the CSPs by updating domains dynamically and rendering such infeasibilities explicit.

We define a CSP infeasible simply when a feasible solution using the assigned domains cannot be found. The easiest infeasibility condition to check is when a variable of the CSP remains with an empty domain as it cannot receive any value.

Our algorithms, which we have called respectively $\mathcal{A}'$, $\mathcal{A}''$ and $\mathcal{A}'''$, are presented in succession where we progressively add ingredients to a simple initial function called $\mathcal{A}$.

A by-product of these algorithms is what we have called the *largest value in the domains* denoted respectively as $\ell_{\mathcal{A}}$, $\ell_{\mathcal{A}'}$, $\ell_{\mathcal{A}''}$ and $\ell_{\mathcal{A}'''}$. Let $\ell_{D_j} = max_{\{\ell \in D_j\}}\ell$ be the maximum value in D_j; we denote as $\ell_{\mathcal{A}}$ the maximum ℓ_{D_j} computed among all the domains D_j of a given CSP handled by $\mathcal{A}$. Moreover, let $\ell_{\mathcal{A}'}$, $\ell_{\mathcal{A}''}$ and $\ell_{\mathcal{A}'''}$ be accordingly defined.

The mapping between a CSP and a coloring problem allows us to state that $\ell_{\mathcal{A}'}$, $\ell_{\mathcal{A}''}$ and $\ell_{\mathcal{A}'''}$ are lower bounds on the chromatic number of G.

We recall that with W we denote a maximal clique and that $|W|$ is a valid lower bound on the chromatic number. Moreover, for the sake of completeness, we recall the definition of *precoloring* of a graph G. Given an integer $k \geq 2$, a *k-coloring* is a function $f : V \to \{1, 2, \ldots, k\}$ such that $f(i) \neq f(j)$ for each $(i, j) \in E$. Given an integer $k \geq 2$, a graph G with $|V| \geq k$, a vertex subset $V' \subseteq V$, a *precoloring* (Kratochvil, 1993) is a proper k-coloring of the subgraph induced by V', and k is called the *color bound* of the precoloring. In order to give an example, a colored clique W is a precoloring of G and $|W|$ is its color bound, i.e., a colored clique W is a $|W|$-precoloring of G.

The first algorithm we present, namely $\mathcal{A}$, finds a starting CSP as follows. It begins from a $|W|$-precoloring of G given by a colored maximal clique W (this coloring can be fixed (Sewell, 1996)).

Define the *neighborhood* of W as $N(W) = \{j \in V \backslash W | (i, j) \in E, i \in W, \}$. Assign domains to the nodes in $N(W)$ as follows: first set $D_i = \{1, \ldots, |W|\}$ for each node $i \in N(W)$; then for each D_i delete colors already assigned to the nodes in W adjacent to i.

Now, the subgraph $W \cup N(W)$ is associated with a CSP where variables X_i with $i \in W$ are associated with a domain $|D_i| = 1$ (represented by the color assigned to i), and variables X_i with $i \in N(W)$ are associated with domains which are proper subsets of $\{1, \ldots, |W|\}$. Trivially $\ell_{\mathcal{A}} = |W|$.

As can be inferred, although we have defined $\mathcal{A}$ starting from W, $\mathcal{A}$ can be executed from a generic precolored set $W' \subseteq V$, $\ell_{\mathcal{A}}$ being its color bound, just by assigning domains $D_i = \{1, \ldots, \ell_{\mathcal{A}}\}$ for each node $i \in N(W')$ and, for each D_i deleting colors already assigned to the nodes in W' adjacent to i (as will be described later on).

Denote with G_{NW} and G_N the subgraphs induced by $W \cup N(W)$ and $N(W)$, respectively.

Proposition 2.1. *$\mathcal{A}$ renders the domains in G_N not empty, i.e., $|D_i| \geq 1$ for each node $i \in N(W)$.*

Proof: We prove the proposition by contradiction. Assume $|D_i| = 0$ for a node i in $N(W)$. Thus i has to be adjacent to all the nodes in W, making W not maximal. Hence $|D_i| \geq 1$. □

In our scenario, the precoloring given by a colored clique can be naturally enlarged whenever at least one node $i \in N(W)$ has $|D_i| = 1$ (see Proposition 2.1). Indeed, if such an i exists it is forced to assume the unique color left in its domain D_i, enlarging the current precolored set W and making room for a further propagation to its neighboring nodes. In order to carry out this occurrence we use $\mathcal{A}$ as a subroutine of a new algorithm which we have called $\mathcal{A}'$, described as follows. $\mathcal{A}'$ receives in input a $|W|$-precoloring of G and

propagates constraints. This can be done dynamically for all nodes i for which $|D_i|$ becomes equal to one during the search. When $\mathcal{A}'$ terminates, it delivers a CSP and a new precolored set $W' \supseteq W$.

Algorithm $\mathcal{A}'$

1. Start from a precolored set $W' = W$; let $\ell_{\mathcal{A}'} = |W|$ be the color bound of this precoloring.
2. Execute $\mathcal{A}$ starting from W'.
3. If there exists a node $i \in N(W')$ with $|D_i| = 1$, then the precolored set is enlarged to $W' = W' \cup \{i\}$ where i is assigned the unique color left in its domain, otherwise `stop`.
4. Set $D_k = \{1, \ldots, \ell_{\mathcal{A}'}\}$ for each node $k \in N(W')$ which is not yet assigned any domain; delete the color assigned to i from D_k for each $k \in N(W')$.
5. If a D_k (see Line 4) becomes empty, then `stop`; otherwise go to Line 3.

The rationale behind $\mathcal{A}'$ is the following: the algorithm tries to iteratively enlarge the precolored set W', initially equal to W (see Line 1), as much as possible, exploiting the nodes forced to assume the unique color left in a domain with cardinality equal to one (see Line 3). Line 4 allows the correct iteration of $\mathcal{A}'$: in fact when W' is enlarged, new nodes in $N(W')$ must be assigned a domain. The whole process terminates either when none of the nodes in $N(W')$ has a domain with cardinality equal to one, or when at least one domain is emptied (see Lines 3 and 5, respectively).

$\mathcal{A}'$ offers an immediate result: whenever a node in $N(W')$ remains with an empty domain (see Line 5) the algorithm stops, as it has found an infeasibility condition. In fact, the CSP obtained cannot have a feasible solution, and, consequently, a $|W|$-coloring does not exist for the subgraph induced by $W' \cup N(W')$. This allows the lower bound to be raised from $|W|$ to $|W| + 1$. Motivated by this initial result we extend the approach in order to get better lower bounds.

$\mathcal{A}'$ uses W as a starting propagation set. In the sequel we denote W as PS_1, i.e., propagation set 1. The idea is to give more power to the constraint propagation performed by $\mathcal{A}'$ by means of an additional propagation node set, namely PS_2. The following is a description of our new approach. Find PS_1 and define a node set PS_2. Denote with G_{PS_1} and G_{PS_2} the graphs induced by PS_1 and PS_2, respectively. Obviously, by definition of PS_1, $\chi(G_{PS_1}) = |W| = |PS_1|$. Let $\ell_{\mathcal{A}''} = max\{|PS_1|, \chi(G_{PS_2})\}$. Execute $\mathcal{A}'$ using as a starting set $W' = PS_1 \cup PS_2$ for each different optimal coloring of PS_2. If $\mathcal{A}'$ at each run returns at least one empty domain, then $\ell_{\mathcal{A}''}$ colors are not sufficient for G and at least $\ell_{\mathcal{A}''} + 1$ colors are needed.

The main difficulty in such an approach is that while colors in PS_1 can be fixed, we cannot fix any colors in PS_2 as we are attempting to find a lower bound and cannot neglect any solution. This means that choosing PS_2 at random can led to difficult computation as we have to enumerate all the optimal colorings of G_{PS_2}.

The easiest way to simplify this computation is to let PS_2 be formed initially by a single node. In this case $|PS_1| = |W| \geq \chi(G_{PS_2}) = 1$ and then $\ell_{\mathcal{A}''} = max\{|PS_1|, \chi(G_{PS_2})\} = |W|$ and the sketch is the following. Find a clique W and select a node $i \in V \backslash W$. Execute $\mathcal{A}'$ $\ell_{\mathcal{A}''}$-times, ($\ell_{\mathcal{A}''}$ being all the different optimal colorings of PS_2) using as a starting set $W' = W \cup \{i\}$ and assigning a different color to i (from 1 to $|W|$) each time. If $\mathcal{A}'$,

at each run, returns at least one empty list, then $|W|$ colors are not sufficient for G and at least $|W|+1$ colors are needed. We denote this algorithm as $\mathcal{A}''$ and its description follows.

Algorithm $\mathcal{A}''$

1. Color a maximal clique W in input.
2. Number nodes in $V\backslash W$ from 1 to $|V\backslash W|$.
3. Examine the numbered nodes one by one starting from $i=1$.
4. Set $\ell_{\mathcal{A}''}=|W|$.
5. From the set $W'=W\cup\{i\}$ execute $\mathcal{A}'$ $\ell_{\mathcal{A}''}$-times, assigning at each run a different color to i starting from 1 to $\ell_{\mathcal{A}''}$.
6. If *for each* color assignment of i there exists at least one domain which becomes empty, then $\ell_{\mathcal{A}''}$ colors are not sufficient; $\ell_{\mathcal{A}''}=\ell_{\mathcal{A}''}+1$; go to Line 5; otherwise go to Line 7.
7. If $i=|V\backslash W|$ then `stop`; otherwise i is removed from W' and the node $i+1$ is considered, i.e., $i=i+1$; go to Line 5.

Note that when $\mathcal{A}''$ finds an infeasibility, $\ell_{\mathcal{A}''}$ is increased and Line 5 restarts from scratch with that node (see Line 6) invoking $\mathcal{A}'$ in order to find a further infeasibility. Moreover, we have designed $\mathcal{A}''$ so as to examine all the nodes in $V\backslash W$. Note again that differently from $\mathcal{A}'$, which was able to increase the lower bound by only one, $\mathcal{A}''$ does not inherit this limitation (see experimental results in Section 4). In Subsection 4.3 an example of the execution of $\mathcal{A}''$ is given.

$\mathcal{A}''$ suggests a further refinement in the application of the proposed technique if we let the propagation set PS_2 be formed by two adjacent nodes belonging to $V\backslash W$. We call this algorithm $\mathcal{A}'''$.

Algorithm $\mathcal{A}'''$

1. Color a maximal clique W in input.
2. Number edges in the subgraph $G_{V\backslash W}$.
3. Examine these edges one by one.
4. Set $\ell_{\mathcal{A}'''}=|W|$, and let (i,j) be the current edge to be examined.
5. From the set $W'=W\cup\{i,j\}$ execute $\mathcal{A}'$ for each possible $\ell_{\mathcal{A}'''}$-coloring assignable to (i,j).
6. If *for each* color assignment there exists at least one domain which becomes empty, then $\ell_{\mathcal{A}'''}$ colors are not sufficient; $\ell_{\mathcal{A}'''}=\ell_{\mathcal{A}'''}+1$; go to Line 5; otherwise go to Line 7.
7. If all the edges have been examined then `stop`; otherwise (i,j) is removed from W' and the successive edge is considered; go to Line 5.

Also $\mathcal{A}'''$ receives in input a maximal clique W and delivers a lower bound $\ell_{\mathcal{A}'''}$. As a direct consequence of how $\mathcal{A}'''$ has been designed, it requires more computational time than $\mathcal{A}'$ and $\mathcal{A}''$, while achieves a better lower bound, i.e., $\ell_{\mathcal{A}'''}\geq\ell_{\mathcal{A}''}\geq\ell_{\mathcal{A}'}$ (see the complexity analysis in the next subsection and the experimental analysis in Section 4).

2.1. *Analysis of the complexity*

The algorithm $\mathcal{A}$ has a worst case complexity $O(|E|)$: indeed, for each edge (i, j) such that $i \in W$ and $j \in N(W)$ it executes exactly one color deletion from D_j in $O(1)$. Recalling that $E(G_{V\setminus W})$ is the edge set induced by $V\setminus W$, we can decrease the complexity to $O(|E(G_{V\setminus W})|)$.

$\mathcal{A}'$ runs in $O(|E|)$. In fact, the only difference between $\mathcal{A}'$ and $\mathcal{A}$ is the following: when a node i remains with a unitary domain, it enlarges the current precolored set W' and deletes the color of i from D_k, where $k \in N(W')$ and $(i, k) \in E$, with a complexity $O(1)$. Moreover, note that each edge is considered at most once.

$\mathcal{A}''$ executes $\mathcal{A}'$ for each node in $V\setminus W$ at most $\ell_{\mathcal{A}''}$-times, and Line 6 can re-iterate the process at most $(\ell_{\mathcal{A}''} - |W|)$-times. Thus, its complexity is $O(\ell_{\mathcal{A}''} * |V\setminus W| * |E| * (\ell_{\mathcal{A}''} - |W|))$ which is bounded by $O(|V|^3 * |E|)$.

$\mathcal{A}'''$ executes $\mathcal{A}'$ for each edge in $G_{V\setminus W}$ at most $\frac{\ell_{\mathcal{A}'''} * (\ell_{\mathcal{A}'''} - 1)}{2} - \ell_{\mathcal{A}'''}$-times, i.e., all the possible combinations of $\ell_{\mathcal{A}'''}$ colors on one edge (the first fractional term) less the monochromatic occurrences which are not admissible (the second term). Moreover, as for the case of $\mathcal{A}''$, Line 6 can re-iterate the process at most $(\ell_{\mathcal{A}'''} - |W|)$-times. Thus, its complexity is $O((\frac{\ell_{\mathcal{A}'''} * (\ell_{\mathcal{A}'''} - 1)}{2} - \ell_{\mathcal{A}'''}) * |E(G_{V\setminus W})| * |E| * (\ell_{\mathcal{A}'''} - |W|))$ which is bounded by $O(|V|^3 * |E|^2)$.

2.2. *Consistency property*

Now, we point out a further insight on our constrained propagation algorithms; in particular we show how their properties can be used to extend the approach in order to achieve a *feasible* solution for the *whole* graph.

Note that when $\mathcal{A}'$, $\mathcal{A}''$ and $\mathcal{A}'''$ terminate their execution, i.e., when they are not able to detect a further infeasibility, the subgraph $W' \cup N(W')$ is associated with a CSP with a number of variables X_i which can be quite large. More precisely, besides the values $\ell_{A'}$, $\ell_{A''}$ and $\ell_{A'''}$, it is important to know how large the CSP for which we are not able to detect infeasibility can be. It could even happen that this CSP covers the whole graph, i.e., $W' \cup N(W') = V$, or most of it. This is a fundamental issue if one wants to seek for a feasible coloring of the whole graph when the lower bound algorithms stop. The main result supporting this idea relies on edge consistency.

Definition 2.1 (Mackworth, 1976). A constraint graph is edge consistent if and only if for each of its edges (i, j) and for any value $x \in D_i$, there is a value $y \in D_j$ that satisfies the binary constraint between i and j.

An analysis of how we have designed our algorithms shows that:

Theorem 2.1. *When $\mathcal{A}'$ ($\mathcal{A}''$, $\mathcal{A}'''$) finishes processing a CSP where infeasibility has not been proved this CSP is edge consistent.*

Proof: Let (i, j) be a generic edge in the constraint graph of such a CSP. Domains D_i and D_j are not empty as we have not found any infeasibility, thus it is always possible to assign

a color to both i and j. Whatever the choice of color is for i, j can be assigned a different color (note that the same holds if we start assigning a color to j). In fact, if $|D_j| \geq 2$ then it is trivially true. Whether $|D_j| = 1$, the color left for j was previously deleted from D_i and thus will not be the same as that assigned to i. □

The latter result is used as a starting point of an iterative process extending the CSP to the whole graph maintaining edge consistency. In the next section, we give details of such an extension providing an algorithm based on constraint propagation, to compute a feasible coloring for the whole graph.

3. Extending CSPs maintaining edge consistency

As mentioned in the last subsection, we will use the largest CSP for which it has not been possible to prove infeasibility as a starting point of the process we describe in this section, which will lead to a feasible coloring for G.

Let this largest CSP be formed by a node set W' which is assigned domains with cardinality equal to one (i.e., it is a precoloring) and a set of neighboring nodes $N(W')$ which is assigned domains with cardinality greater than one (note that this is the current structure of a CSP returned by our algorithms). In the sequel we will denote with $G_{N'}$ the subgraph induced by $N(W')$ and with $G_{NW'}$ the subgraph induced by $W' \cup N(W')$. Moreover, as this process can be applied by using the largest CSP returned by either $\mathcal{A}'$ or $\mathcal{A}''$ or $\mathcal{A}'''$, one can execute the next algorithm after one of the three above mentioned algorithms has run. In order to avoid redundancies, we describe the algorithm starting from the output of $\mathcal{A}'''$, the description being identical for $\mathcal{A}'$ and $\mathcal{A}''$. We denote this algorithm as $\mathcal{F}'''$ (accordingly, we will denote as $\mathcal{F}'$ and $\mathcal{F}''$ the same algorithm when applied after $\mathcal{A}'$ and $\mathcal{A}''$, respectively). The main result we have used to engineer our algorithm is the following:

Theorem 3.1 (Freuder, 1989)**.** *If a constraint graph is a tree and if it is edge consistent, then it admits backtrack free solutions.*

We recall that a constraint graph admits *backtrack free* solutions if, given any order of the variables, it is possible to assign values to variables one by one (respecting constraints and domains), without changing previous assignments.

Note that as $G_{N'}$ is edge consistent a first result can be read out directly from Theorem 3.1:

Theorem 3.2 *If $G_{N'}$ is a forest, then $G_{NW'}$ has chromatic number $\ell_{\mathcal{A}'''}$.*

Proof: $G_{N'}$ is edge consistent and by hypothesis is a forest. Hence, from Theorem 3.1, a backtrack free coloring can be computed exploiting the domains assigned. As the latter are formed by colors at most equal to $\ell_{\mathcal{A}'''}$, $G_{N'}$ is $\ell_{\mathcal{A}'''}$-colorable and $G_{NW'}$ has chromatic number $\ell_{\mathcal{A}'''}$ as well. □

In general, however, $G_{N'}$ is not a forest and the problem of determining whether the precoloring given by the colored set W' is extendible to its neighborhood is $\mathcal{NP}$-complete

(Kratochvil, 1993). Thus we have designed an algorithm, namely *SPLIT*, which, starting from $G_{N'}$, finds a forest $F_{N'}$ by deleting some edges in $G_{N'}$. The forest found remains edge consistent, as domains have not been changed, and one can easily compute a backtrack free solution for $F_{N'}$ (see Theorem 3.1). However, it may happen that the resulting coloring returns monochromatic couples of nodes which were adjacent in $G_{N'}$, thus producing a solution for $F_{N'}$ which is not feasible for $G_{N'}$.

This possible drawback is circumvented by means of a function which we have denoted as REC: for each edge (i, j) deleted by *SPLIT*, *REC* makes $D_i \cap D_j = \emptyset$ and $|D_i| \geq 2, |D_j| \geq 2$. This allows the algorithm to return an edge consistent forest, where each coloring for $F_{N'}$ is also proper for $G_{N'}$. However, this task may require the introduction of a number of additional colors denoted as Δ. In the following, we describe the functions *SPLIT* and *REC*.

Algorithm *SPLIT*

1. Initialize $\Delta = 0$.
2. Arbitrarily choose a node i of $G_{N'}$ and label it permanent;
3. Label the nodes of its adjacency list temporary.
4. While all the nodes of $G_{N'}$ are not labeled permanent do:
  4.1. Arbitrarily choose a temporary node i and label it permanent.
  4.2. For each node j adjacent to i that is not yet labeled permanent do:
    4.2.1 If j has already been labeled temporary ((i, j) makes a cycle in $G_{N'}$) then:
      4.2.1.1. Delete the edge (i, j).
      4.2.1.2. $REC(i, j)$.
    4.2.2. Else label the node j temporary ((i, j) does not make a cycle in $G_{N'}$).
5. $F_{N'} = G_{N'}$.

Algorithm $REC(i, j)$

1. Mark (i, j) as visited; let D be the intersection of D_i and D_j and remove common colors from these two domains.
2. If $|D_i| \geq 2$ and $|D_j| \geq 2$ then merge D with D_i or D_j arbitrary.
3. If $|D_i| \geq 2 (|D_j| \geq 2)$ and $|D_j| < 2 (|D_i| < 2)$ then merge D with D_j (merge D with D_i).
4. If $|D_i| < 2$ and $|D_j| < 2$ then partition D into two subsets D' and D'' such that $|D_i \cup D'| \geq 2$ and $|D_j \cup D''| \geq 2$ and merge D' with D_i and D'' with D_j; if this is not possible then
  4.1. Partition D into two subsets D' and D'' such that both $D_i \cup D'$ and $D_j \cup D''$ have possibly at least a color in the set $\{1, \ldots, \ell_{\mathcal{A}'''}\}$, and merge D' with D_i and D'' with D_j.
  4.2. While $|D_i| < 2$ do
    4.2.1. $\Delta = \Delta + 1$.
    4.2.2. $D_i = D_i \cup \{\ell_{\mathcal{A}'''} + \Delta\}$; moreover, let $D_s = D_s \cup \{\ell_{\mathcal{A}'''} + \Delta\}$ for each node s which is not in a visited edge.
    4.2.3. If $|D_j| < 2$ then

4.2.3.1. $\Delta = \Delta + 1$.

4.2.3.2. $D_j = D_j \cup \{\ell_{\mathcal{A}'''} + \Delta\}$; moreover, let $D_s = D_s \cup \{\ell_{\mathcal{A}'''} + \Delta\}$ for each node s which is not in a visited edge.

4.3. While $|D_j| < 2$ do

4.3.1. $\Delta = \Delta + 1$.

4.3.2. $D_j = D_j \cup \{\ell_{\mathcal{A}'''} + \Delta\}$; moreover, let $D_s = D_s \cup \{\ell_{\mathcal{A}'''} + \Delta\}$ for each node s which is not in a visited edge.

/*Although Lines 4.2.3, 4.2.3.1 and 4.2.3.2 are a repetition of Lines 4.3, 4.3.1 and 4.3.2, they play and important role in balancing the composition of the domains D_i and D_j when Line 4 is executed. If implemented differently, e.g., these lines are removed, in Line 4.2 the domain D_i will be updated until $|D_i| \geq 2$, while D_j will remain unchanged until Line 4.3. This in general worsens the quality of the coloring found.*/

As the algorithm *SPLIT* transforms $G_{N'}$ into a forest $F_{N'}$ that is edge consistent, it follows (directly from Theorem 3.1):

Theorem 3.3. *It is always possible to find an admissible backtrack free coloring for* $F_{N'}$.

Theorem 3.4. *The admissible coloring found for* $F_{N'}$ *is also admissible for* $G_{N'}$.

Proof: We prove the theorem showing that the edges deleted from $G_{N'}$ can added to $F_{N'}$ without modifying the coloring of the latter. The edges (i, j) of $G_{N'}$ removed by *SPLIT* in order to obtain $F_{N'}$ have domains D_i and D_j modified by *REC* such that $D_i \cap D_j = \emptyset$. Thus, the assignment of colors in $F_{N'}$ returns necessarily different colors for i and j. This property holds for all the edges (i, j) removed from $G_{N'}$, thus they can be added in $F_{N'}$ without modifying its admissible coloring. □

Let $\ell^1_{\mathcal{F}'''}$ be the maximum value in the domain assigned to $G_{NW'}$ by $\mathcal{F}'''$, which represents an upper bound on $\chi(G_{NW'})$.

In the sequel we describe how to enlarge the CSP. Recall that, in Section 2 we defined the algorithm $\mathcal{A}$ starting from the precolored set W'. The process described for $G_{NW'}$ can be applied for a subgraph induced by a generic neighborhood of W'. For this purpose, let us define $N^k(W') = N(N^{k-1}(W'))$ where $N^0(W') = W'$, and denote with $G_{N^{k-1}W'}$ and $G_{N'^k}$ the subgraphs induced, respectively, by $\cup_{j=0}^{k-1} N^j(W')$ and $N^k(W')$. Now, we introduce the generalization of $\mathcal{A}$ denoted as $\mathcal{A}_k$. Start from the precolored set $\cup_{j=0}^{k-1} N^j(W')$, where $\ell^{k-1}_{\mathcal{F}'''}$ is its color bound, and assign to each node in $N^k(W')$ a domain $\{1, \ldots, \ell^{k-1}_{\mathcal{F}'''}\}$. For each node $j \in N^k(W')$ delete from D_j the colors assigned to nodes $i \in \cup_{j=0}^{k-1} N^j(W')$ adjacent to j.

Algorithm $\mathcal{A}_k$

1. Assign to each vertex $j \in N^k(W')$ a domain $D_j = \{1, \ldots, \ell^{k-1}_{\mathcal{F}'''}\}$.

2. For each node $j \in N^k(W')$ delete from D_j the colors already assigned to nodes in $N^{k-1}(W')$ adjacent to j.

Once $\mathcal{A}_k$ has run, if $G_{N'^k}$ is edge consistent then *SPLIT* is invoked on such a subgraph and, successively, an admissible coloring for the forest $F_{N'^k}$ so obtained is found. As previously proved for $k = 1$, the coloring found for $F_{N'^k}$ is admissible for $G_{N'^k}$.

However, it can happen that $G_{N'^k}$ is not edge consistent. For this case, we have designed an algorithm, namely *AEC*, which makes $G_{N'^k}$ edge consistent, rendering all the domains in $G_{N'^k}$ not empty and each edge (i, j) of $G_{N'^k}$ such that $D_i \cap D_j = \emptyset$.

Algorithm *AEC*

1. $\Delta = 0$;
for each edge (i, j) of $G_{N'^k}$ do

1.1. Mark (i, j) as visited; let D be the intersection of D_i and D_j and remove common colors from these two domains.

1.2. If neither D_i nor D_j is empty, merge D with one of them arbitrarily. (i, j) is now edge consistent.

1.3. If only one of D_i or D_j is empty, assign it the value of D. (i, j) is now edge consistent.

1.4. If both are empty and $|D| \geq 2$, partition D into D' and D'' such that $|D_i \cup D'| \geq 1$ and $|D_j \cup D''| \geq 1$, and merge D_i with D' and D_j with D''. (i, j) is now edge consistent.

1.5. If both are empty and $|D| \leq 1$ then

1.5.1. Merge D with D_i or D_j arbitrarily.

1.5.2. If D_i is empty then
$\Delta = \Delta + 1$.
$D_i = D_i \cup \{\ell_{\mathcal{F}'''^{k-1}} + \Delta\}$; moreover, let $D_s = D_s \cup \{\ell_{\mathcal{F}'''^{k-1}} + \Delta\}$ for each node s which is not in a visited edge.

1.5.3. If D_j is empty then
$\Delta = \Delta + 1$.
$D_j = D_j \cup \{\ell_{\mathcal{F}'''^{k-1}} + \Delta\}$; moreover, let $D_s = D_s \cup \{\ell_{\mathcal{F}'''^{k-1}} + \Delta\}$ for each node s which is not in a visited edge.

(i, j) is now edge consistent.

Note that, although *AEC* seems very similar to *REC*, there is a main difference between them: while *REC* makes *only* the domains D_i and D_j in input non intersecting with $|D_i| \geq 2$ and $|D_j| \geq 2$, *AEC* processes *all* the edges of $G_{N'^k}$ making them non intersecting with $|D_i| \geq 1$ and $|D_j| \geq 1$. This assures that, although $G_{N'^k}$ is generally not a forest, it admits backtrack free solutions. Thus:

Theorem 3.5. *It is always possible to find an admissible backtrack free coloring for* $G_{N'^k}$.

Let $\ell^k_{\mathcal{F}'''}$ be the color bound obtained for $G_{N^k W'}$. As can be inferred, $\mathcal{F}'''$ stops when a feasible coloring for the whole graph has been found. We denote with $\ell_{\mathcal{F}'''}$ such an upper bound on $\chi(G)$. In Subsection 4.3 we provide an example of the execution of $\mathcal{F}'''$.

3.1. Analysis of the complexity

SPLIT runs in $O(|V|^2)$. It examines, at each iteration, one node in the graph and labels its neighbors. When it detects a cycle, it invokes *REC* which runs in $O(|V|)$. Thus, the overall worst case complexity is $O(|V|^3)$.

AEC has a worst case complexity $O(|E| * |V|)$. It visits each edge once, and each time computes the intersection of the domains, merging this intersection and eventually adding more colors to the domains.

4. Experimental analysis

After the algorithm description in Sections 2 and 3, we devote this section to the experimental analysis where we report results obtained by $\mathcal{A}'$, $\mathcal{A}''$, $\mathcal{A}'''$ and $\mathcal{F}'$, $\mathcal{F}''$, $\mathcal{F}'''$.

4.1. Computer and implementation environments

In order to allow comparisons between algorithms which have been implemented on different machines, it is necessary to know the approximate speeds of the computers involved.

For this purpose, we report in Table 1 the values obtained for the DIMACS Machine benchmarks by our platform (see Table 2). These benchmarks are available at the DIMACS ftp site (ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/volume/machine), and consist of a benchmark program (DFMAX) for solving the maximum clique problem and five benchmark graphs (r100.5, r200.5, r300.5, r400.5, r500.5).

4.2. Computational results

The experimental results are divided into two different parts. The first set of experiments reports results on random graphs: we have considered generic random graphs having from

Table 1. CPU time obtained for the DIMACS Machine benchmarks.

Graph	r100.5	r200.5	r300.5	r400.5	r500.5
CPU	0.00	0.07	0.64	4.00	15.46

User time in seconds.

Table 2. Experiment environment characteristics.

Machine: Workstation Digital Alpha Model 1000A 5/500
RAM: 256 Mb
Cache memory: 8 Mb
Computer language: 'C'
Compiler: standard Digital
Operating system: Unix 40B

100 to 700 nodes with an edge probability of 0.5 and 0.7, and triangle free random graphs having from 100 to 700 nodes. These latter graphs are generated by applying a modified version of *SPLIT* (see Section 3) which eliminates only cycles of length three. It starts from a generic random graph with density 0.5 and whenever it finds an edge forming a triangle it deletes this edge. We believe that triangle free graphs represent a particular class of instances where our lower bound algorithms are capable of giving rise to a high number of infeasibilities and so we can have additional information on their performance.

Besides random graphs, we experimented with benchmark instances. Among the well known benchmark graphs available at (ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/) we have selected those having a difference $\chi(G) - \omega(G)$ greater than zero. We remark that this choice is not restrictive as we want to verify whether the proposed lower bound algorithms are able to improve the clique bound.

For each of the tested graph classes, we report results in three different tables. A first table collects results related to some structural characteristics of the graphs. A second table shows the solution quality achieved by $\mathcal{A}'$, $\mathcal{A}''$, $\mathcal{A}'''$ and $\mathcal{F}'$, $\mathcal{F}''$, $\mathcal{F}'''$. Finally, the last table reports the running times.

As mentioned in the introduction, exact coloring codes rely on the cardinality of a maximum clique as lower bound on the chromatic number. For this reason, we have included $\mathcal{A}'$, $\mathcal{A}''$ and $\mathcal{A}'''$ as subroutines of an algorithm based on dynamic programming for the maximum clique problem, executing them only on incumbent maximal cliques with cardinality not smaller than the current best one found.

This means that, in triangle free graphs, we have tested the algorithms starting from each edge. This in some sense motivates the higher running times of the algorithms on the latter graphs than on the corresponding general random ones (see Tables 7 and 8).

Finally, we remark an important implementation detail concerning the lower bound algorithms. $\mathcal{A}'$ is executed only on maximum cardinality cliques as its contribution can be to increase the clique bound by only one. In fact, suppose we have a maximal clique of cardinality $\omega(G) - 1$: $\mathcal{A}'$ can return a lower bound at most equal to $\omega(G)$, and when we will find next a maximum clique, the contribution given by $\mathcal{A}'$ will be neglected. These considerations do not hold for $\mathcal{A}''$ and $\mathcal{A}'''$: indeed, to get the best lower bound achievable, we executed the latter algorithms starting from each maximal clique found with the limitation described above.

Experimental data, and in particular coloring solutions have been made available in the public domain at `http://rosd.sta.uniroma1.it`. We now give details of our findings in the following two subsections.

4.2.1 Random graphs. Computational results on random graphs are structured as follows: Tables 3 and 4 report the characteristics of the graphs involved in the computation, pointing out the following values:

1. The minimum and maximum number of neighborhoods of a generic starting clique (denoted min_k and max_k respectively).
2. The cardinality of the maximum clique ($\omega(G)$).

Table 3. Characteristics of random graphs.

Type	min_k	max_k	$\omega(G)$
100.5	1	2	8.0
100.7	1	2	10.0
200.5	1	2	14.0
200.7	1	2	17.0
300.5	1	2	24.4
300.7	1	2	26.6
400.5	1	2	36.8
400.7	1	2	40.2
500.5	1	2	42.8
500.7	1	2	46.2
600.5	1	2	49.2
600.7	1	2	52.0
700.5	1	2	55.0
700.7	1	2	58.2

Statistics are average of 5 instances.

Table 4. Characteristics of triangle free random graphs.

Type	min_k	max_k	$\omega(G)$
TF100	2	3	2.0
TF200	2	3	2.0
TF300	2	3	2.0
TF400	2	3	2.0
TF500	2	3	2.0
TF600	2	3	2.0
TF700	2	3	2.0

Statistics are average of 5 instances.

Tables 5 and 6 report results inherent to the following parameters:

1. The cardinality $|W|$ of the maximal clique used as input for both $\mathcal{A}''$ and $\mathcal{A}'''$ and giving the best lower bound (see the next point 4).
2. The number of neighborhoods (denoted as k_W) of the starting clique W which gives the best lower bound for both $\mathcal{A}''$ and $\mathcal{A}'''$ (see the next point 4).
3. The index k_{PS_2} of the neighborhood where the second set of propagation belongs to.
4. The lower bounds $\ell_{\mathcal{A}'}$, $\ell_{\mathcal{A}''}$ and $\ell_{\mathcal{A}'''}$.
5. The upper bounds $\ell_{\mathcal{F}'}$, $\ell_{\mathcal{F}''}$ and $\ell_{\mathcal{F}'''}$.

Table 5. ℓ-values for random graphs.

Type	$\|W\|$	k_W	k_{PS_2}	$\ell_{\mathcal{A}'}$	$\ell_{\mathcal{A}''}$	$\ell_{\mathcal{A}'''}$	$\ell_{\mathcal{F}'}$	$\ell_{\mathcal{F}''}$	$\ell_{\mathcal{F}'''}$
100.5	7.0	2	2	9.0	10.0	12.0	16.4	16.2	16.0
100.7	8.0	2	2	11.0	12.0	14.0	18.2	18.0	18.0
200.5	12.0	2	2	15.0	17.0	18.0	27.4	27.2	27.0
200.7	15.0	2	2	18.0	20.0	21.0	30.4	30.2	30.0
300.5	20.2	2	2	25.2	26.4	27.0	38.8	38.6	37.8
300.7	24.2	2	2	27.2	27.2	29.2	42.0	41.8	41.6
400.5	34.4	2	2	37.8	38.2	40.0	47.0	46.8	46.4
400.7	38.0	2	2	41.2	42.0	45.4	51.2	50.8	50.6
500.5	40.0	2	2	43.4	44.0	46.8	53.8	53.0	52.2
500.7	44.0	2	2	47.2	49.0	50.0	58.0	57.2	56.4
600.5	46.2	2	2	50.2	53.8	54.0	64.2	62.6	60.8
600.7	50.0	2	2	53.0	56.2	58.4	70.0	68.4	66.8
700.5	52.0	2	2	56.0	58.0	59.4	68.6	64.8	64.2
700.7	56.4	2	2	59.2	62.0	64.0	74.4	72.2	70.8

Statistics are average of 5 instances.

Table 6. ℓ-values for triangle free random graphs.

Type	$\|W\|$	k_W	k_{PS_2}	$\ell_{\mathcal{A}'}$	$\ell_{\mathcal{A}''}$	$\ell_{\mathcal{A}'''}$	$\ell_{\mathcal{F}'}$	$\ell_{\mathcal{F}''}$	$\ell_{\mathcal{F}'''}$
TF100	2.0	2.0	2.0	3.0	4.0	4.2	10.0	9.2	9.0
TF200	2.0	2.0	2.0	3.0	4.0	4.6	10.2	9.6	9.0
TF300	2.0	2.0	2.0	3.0	4.0	4.6	10.8	9.6	9.2
TF400	2.0	2.0	2.0	3.0	4.0	4.6	11.0	10.8	10.4
TF500	2.0	2.0	2.0	3.0	4.0	4.8	13.8	12.0	11.2
TF600	2.0	2.0	2.0	3.0	4.0	4.8	14.2	12.8	12.6
TF700	2.0	2.0	2.0	3.0	4.0	5.0	18.6	15.4	14.2

Statistics are average of 5 instances.

We recall that the three upper bound values $\ell_{\mathcal{F}'}$, $\ell_{\mathcal{F}''}$ and $\ell_{\mathcal{F}'''}$ are obtained by applying $\mathcal{F}'$, $\mathcal{F}''$ and $\mathcal{F}'''$ from $\ell_{\mathcal{A}'}$, $\ell_{\mathcal{A}''}$ and $\ell_{\mathcal{A}'''}$, respectively.

Tables 7 and 8 report the running times (in seconds) of our algorithms: note that the running times of $\ell_{\mathcal{F}'}$, $\ell_{\mathcal{F}''}$ and $\ell_{\mathcal{F}'''}$ include the time needed to compute the corresponding lower bounds.

Results on generic random graphs show the good performance of the proposed algorithms: $\ell_{\mathcal{A}'''}$ significantly increments $|W|$ and $\omega(G)$ as well. In particular, note that $|W| \leq \omega(G) \leq \ell_{\mathcal{A}'} \leq \ell_{\mathcal{A}''} \leq \ell_{\mathcal{A}'''}$ for all the graphs and, in many cases, the inequality is strict. Moreover, these improved lower bounds can be obtained in a very limited running time especially if compared with the graph sizes.

Table 7. CPU times (in seconds) for random graphs.

Type	CPU $\ell_{\mathcal{A}'}$	CPU $\ell_{\mathcal{A}''}$	CPU $\ell_{\mathcal{A}'''}$	CPU $\ell_{\mathcal{F}'}$	CPU $\ell_{\mathcal{F}''}$	CPU $\ell_{\mathcal{F}'''}$
100.5	0.0	1.2	6.8	0.2	1.4	7.0
100.7	0.0	1.8	9.2	0.4	2.2	9.6
200.5	0.6	2.2	9.0	0.8	2.8	9.6
200.7	0.8	2.8	10.8	1.8	3.6	11.6
300.5	1.0	4.8	9.2	1.8	5.6	10.0
300.7	1.2	7.2	11.4	4.0	12.2	15.4
400.5	1.4	8.2	14.6	4.0	14.2	18.6
400.7	1.8	9.2	18.9	5.2	15.4	20.2
500.5	1.8	10.2	24.8	5.2	15.4	26.0
500.7	2.0	12.4	30.4	11.4	23.8	44.4
600.5	2.2	16.0	23.8	22.0	28.0	35.0
600.7	2.4	17.4	31.8	22.2	39.6	44.0
700.5	2.6	18.6	20.4	32.2	60.8	62.6
700.7	3.0	20.6	34.4	32.6	63.2	66.8

Statistics are average of 5 instances.

Table 8. CPU times (in seconds) for triangle free random graphs.

Type	CPU $\ell_{\mathcal{A}'}$	CPU $\ell_{\mathcal{A}''}$	CPU $\ell_{\mathcal{A}'''}$	CPU $\ell_{\mathcal{F}'}$	CPU $\ell_{\mathcal{F}''}$	CPU $\ell_{\mathcal{F}'''}$
TF100	0.0	1.2	6.8	0.2	1.4	7.0
TF200	0.6	4.2	19.0	4.2	8.6	23.6
TF300	1.0	10.8	39.2	6.8	17.4	45.0
TF400	1.4	18.2	54.8	12.4	29.2	68.2
TF500	1.8	30.2	84.8	14.2	45.4	100.4
TF600	2.2	47.0	123.8	24.2	68.0	146.0
TF700	2.6	68.8	160.4	44.8	106.8	202.6

Statistics are average of 5 instances.

Results on triangle free random graphs give evidence of what we have mentioned before: the running times are higher than those of the corresponding generic random graphs. This is motivated by the fact that the algorithms have to be executed from all the cliques (the edges) of the graphs, thus increasing the running times. However, other implementation policies can give lower computational times if, for example, only a subset of these edges is considered. The quality of the lower bound obtained is very high considering that on the average, $\mathcal{A}'''$ obtains a minimum increment of 2.2 and a maximum increment of 3.0 with respect to the cardinality of the maximum clique $\omega(G) = 2$.

A general behavior of the upper bound algorithms is that the *higher* the lower bound in input, the *lower* the upper bound obtained by $\mathcal{F}$ (see Section 3). Indeed, a solution with a

higher lower bound allows either *REC* or *AEC* to use fewer additional colors as the domains have more colors and the intersection is able to satisfy the new domain composition required. Moreover, our experiments show that the *higher* the lower bound, the *higher* the number of variables of the starting CSP used as input for $\mathcal{F}$.

4.2.2 Benchmarks. In this subsection we report our findings on our experimentation with benchmark graphs. We have considered the following data sets: *Mycielski graphs*, *Queen graphs*, *Petersen graph* and *Flat graphs*, representing benchmarks with $\chi(G) > \omega(G)$ (see Tables 9 and 10). In particular with respect to Queen graphs, we have considered in our data set only the graph subset characterized by this last property.

Tables 11 and 12 contain the values $\ell_{\mathcal{A}'}$, $\ell_{\mathcal{A}''}$, $\ell_{\mathcal{A}'''}$ and $\ell_{\mathcal{F}'}$, $\ell_{\mathcal{F}''}$, $\ell_{\mathcal{F}'''}$ obtained for miscellaneous and Flat benchmarks, respectively. Tables 13 and 14 report the running times of our algorithms on the same graph classes.

Recalling what was done in the previous subsection for triangle free random graphs, we examine nontrivial triangle free benchmarks with increasing chromatic number, starting from a 3-chromatic graph (Petersen graph Gondran and Minoux, 1984). One can see that for the Petersen graph, $\mathcal{A}'$ already finds the optimal solution. However, the Petersen graph is

Table 9. Characteristics of miscellaneous benchmarks.

Type	min_k	max_k	$\omega(G)$
Queen6.6	1	2	6
Queen8.8	1	2	8
Queen9.9	1	2	9
Petersen	2	2	2
Myciel3	2	2	2
Myciel4	2	2	2
Myciel5	2	2	2
Myciel6	2	2	2
Myciel7	2	2	2

Table 10. Characteristics of Flat benchmarks.

Type	min_k	max_k	$\omega(G)$
Flat300_20_0	1	2	11
Flat300_26_0	1	2	11
Flat300_28_0	1	2	11
Flat1000_50_0	1	2	11
Flat1000_60_0	1	2	11
Flat1000_76_0	1	2	11

Table 11. ℓ-values for miscellaneous benchmarks.

Type	$\|W\|$	k_W	k_{PS_2}	$\ell_{\mathcal{A}'}$	$\ell_{\mathcal{A}''}$	$\ell_{\mathcal{A}'''}$	$\ell_{\mathcal{F}'}$	$\ell_{\mathcal{F}''}$	$\ell_{\mathcal{F}'''}$
Queen6.6	6	2	2	7	7	7	7	7	7
Queen8.8	7	2	2	8	8	9	9	9	9
Queen9.9	9	2	2	9	9	10	11	11	11
Petersen	2	2	2	3	3	3	3	3	3
Myciel3	2	2	2	3	4	4	4	4	4
Myciel4	2	2	2	3	4	5	5	5	5
Myciel5	2	2	2	3	4	6	6	6	6
Myciel6	2	2	2	3	4	7	7	7	7
Myciel7	2	2	2	3	4	8	8	8	8

Table 12. ℓ-values for Flat benchmarks.

Type	$\|W\|$	k_W	k_{PS_2}	$\ell_{\mathcal{A}'}$	$\ell_{\mathcal{A}''}$	$\ell_{\mathcal{A}'''}$	$\ell_{\mathcal{F}'}$	$\ell_{\mathcal{F}''}$	$\ell_{\mathcal{F}'''}$
Flat300_20_0	10	2	2	12	13	15	25	24	20
Flat300_26_0	10	2	2	12	13	15	30	29	26
Flat300_28_0	10	2	2	12	13	15	34	32	28
Flat1000_50_0	10	2	2	12	14	17	56	54	50
Flat1000_60_0	10	2	2	12	14	17	66	64	60
Flat1000_76_0	10	2	2	12	14	17	81	80	76

very small sized, and so we experimented with graphs having a larger number of nodes and a higher chromatic number. In particular, a well known class of such triangle free graphs is given by Mycielski (1955). We propose tests for Mycielski graphs of order $i = 3, 4, 5, 6$ and 7. This class is characterized by a chromatic number equal to $i + 1$, where i is the index of the graph. Analyzing our results, it can be noted that $\mathcal{A}'$ finds $\omega(G) + 1$, while $\mathcal{A}''$ finds $\omega(G) + 2$. What is remarkable is that $\mathcal{A}'''$ is able to find a lower bound equal to $\ell_{\mathcal{F}'''}$, thus giving the optimal solution. We believe that this algorithmic approach is very powerful on this graph class since the published branch and bound algorithms are able to provide exact solutions up to a Mycielski of order 5 (Caramia and Dell'Olmo, 2001a; Mehrotra and Trick, 1996; Sewell, 1996), while the best lower bound obtained is 4 given by a mathematical programming approach Mehrotra and Trick (1996). This experimental behavior on Mycielski graphs could be surprising; yet it is also supported by recent analytical results which allow us to compute a lower bound on the chromatic number of Mycielski graphs which is tight up to $i = 8$ (Caramia and Dell'Olmo, 2001b).

Our last comment on computational results is devoted to Flat benchmarks, where $\mathcal{A}'$, $\mathcal{A}''$ and $\mathcal{A}'''$ always improve on the cardinality of the maximum clique. In particular, as can be noted in Table 12, this improvement ranges from 1 to 6. Furthermore, results achieved by $\mathcal{F}'''$ improve on those currently published. For example, $\mathcal{F}'''$ needs 76 colors for a Flat1000_76_0 with respect to the 84 employed in Fleurent and Ferland (1996), and is able

Table 13. CPU times (in seconds) for miscellaneous benchmarks.

Type	CPU $\ell_{\mathcal{A}'}$	CPU $\ell_{\mathcal{A}''}$	CPU $\ell_{\mathcal{A}'''}$	CPU $\ell_{\mathcal{F}'}$	CPU $\ell_{\mathcal{F}''}$	CPU $\ell_{\mathcal{F}'''}$
Queen6.6	0.0	0.0	0.0	0.0	0.0	0.0
Queen8.8	0.0	1.2	1.8	0.0	1.2	1.8
Queen9.9	0.0	2.0	2.4	0.0	2.0	2.4
Petersen	0.0	0.0	0.0	0.0	0.0	0.0
Myciel3	0.0	0.0	0.0	0.0	0.0	0.0
Myciel4	0.0	0.4	1.2	0.0	0.4	1.2
Myciel5	0.0	0.6	2.2	0.0	0.6	2.2
Myciel6	0.0	0.8	2.9	0.0	0.8	2.9
Myciel7	0.0	1.8	5.0	0.0	2.0	5.2

Table 14. CPU times (in seconds) for flat benchmarks.

Type	CPU $\ell_{\mathcal{A}'}$	CPU $\ell_{\mathcal{A}''}$	CPU $\ell_{\mathcal{A}'''}$	CPU $\ell_{\mathcal{F}'}$	CPU $\ell_{\mathcal{F}''}$	CPU $\ell_{\mathcal{F}'''}$
Flat300_20_0	10.0	22.0	30.0	20.8	32.2	40.5
Flat300_26_0	12.2	25.4	31.4	24.2	35.4	41.4
Flat300_28_0	15.4	26.6	32.8	25.4	36.6	42.8
Flat1000_50_0	30.0	50.8	52.8	50.5	82.8	89.6
Flat1000_60_0	32.0	52.8	54.8	62.2	92.6	94.6
Flat1000_76_0	34.0	56.8	58.7	64.9	98.6	98.9

to find a 26-coloring for a Flat1000_26_0 where 31 was the previous best know solution (Morgenstern, 1996).

4.3. Examples and further comments

This subsection has the goal of providing examples on how the lower and upper bound algorithms work, believing that this can help to give further insight on the proposed approaches. We start simulating the execution of $\mathcal{A}''$ on the benchmark graph Mycielski of order 3.

Recall that $\mathcal{A}''$ uses two propagation sets: PS_1 represented by a maximal clique W, and PS_2 represented by a node in $V \backslash W$. We identify the starting maximal clique (i.e., PS_1) as the node set which is assigned underlined colors (e.g., $\underline{1}$ and $\underline{2}$), while the set PS_2 is identified by the node i which is assigned colors in bold (see figure 1). Initially, $W' = PS_1 \cup PS_2 = W \cup \{i\}$. In brackets we have reported the domains assigned to the nodes in $N(W')$: when a domain remains with a unique color, brackets are deleted and the corresponding node is put in W'.

Referring to our algorithm description (see Section 2), we execute $\mathcal{A}''$ starting from Line 5 as the node i forming PS_2 has been chosen.

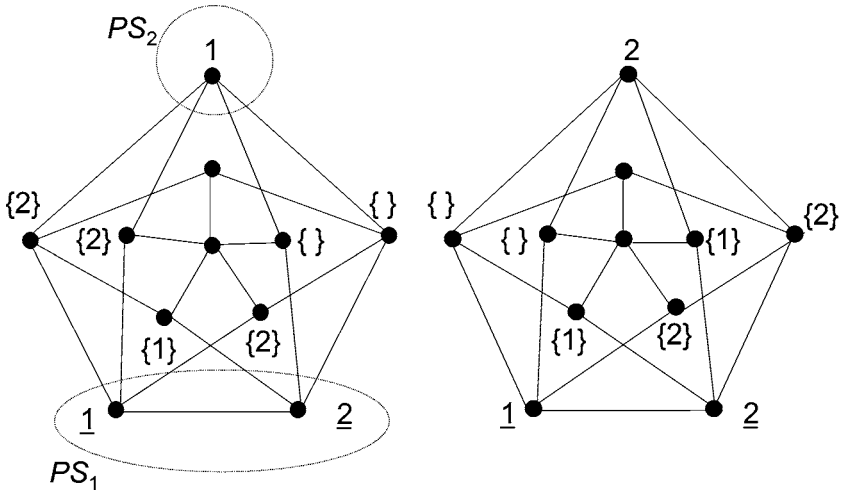

Figure 1. $\mathcal{A}''$ run on Myciel3.

In Figure 1 we report the execution of $\mathcal{A}'$ (see Line 5 of $\mathcal{A}''$) starting from the precolored set $W' = W \cup \{i\}$, where i is assigned color 1. As can be noted, this returns empty domains. Furthermore, figure 1 shows that the same holds when i is assigned color 2. Thus, the lower bound can be raised from 2 to 3 (see Line 6 of $\mathcal{A}''$).

The algorithm restarts from the precolored set $W' = W \cup \{i\}$ where i is assigned color 1 and the domains initially assigned to the nodes in $V \backslash W'$ are enlarged to $\{1, 2, 3\}$ (see Line 6 of $\mathcal{A}''$). Figures 2–4 report the execution of $\mathcal{A}'$, where a domain is emptied (figure 4). The successive restarts of the algorithm, respectively from the precolored set $W' = W \cup \{i\}$ where i is assigned color 2, and from $W' = W \cup \{i\}$ where i is assigned color 3, are symmetric to that in which i is assigned color 1. In both cases, some domains remain empty and the lower bound is improved to 4.

No lower bound improvement is obtainable restarting from $W' = W \cup \{i\}$ with domains enlarged to $\{1, 2, 3, 4\}$, and thus $\mathcal{A}''$ returns $\ell_{\mathcal{A}''} = 4$.

Now, we present an example of the execution of the algorithm $\mathcal{F}$. In particular, in this example we show the behavior mentioned in the computational analysis of the latter algorithm: the *higher* the lower bound in input, the *lower* the upper bound obtained by $\mathcal{F}$. In order to explain this, we use a small graph such as the Petersen graph and the same notation as in the previous example.

Two cases are discussed. The first one refers to the execution of $\mathcal{F}$ starting from the lower bound offered by the cardinality of a maximum clique (we recall that Petersen graph is triangle free and thus $\omega(G) = 2$). The second case refers to the execution of $\mathcal{F}$ starting from a lower bound $\omega(G) + 1$ found by $\mathcal{A}'$. Our goal is to compare the solutions obtained in the two cases in order to show the better efficacy of the algorithm when it uses improved lower bounds.

If we execute $\mathcal{F}$ starting from $\omega(G)$, the first CSP to be considered is the one formed by $W \cup N(W)$, being, respectively, the maximum starting clique and its neighborhood.

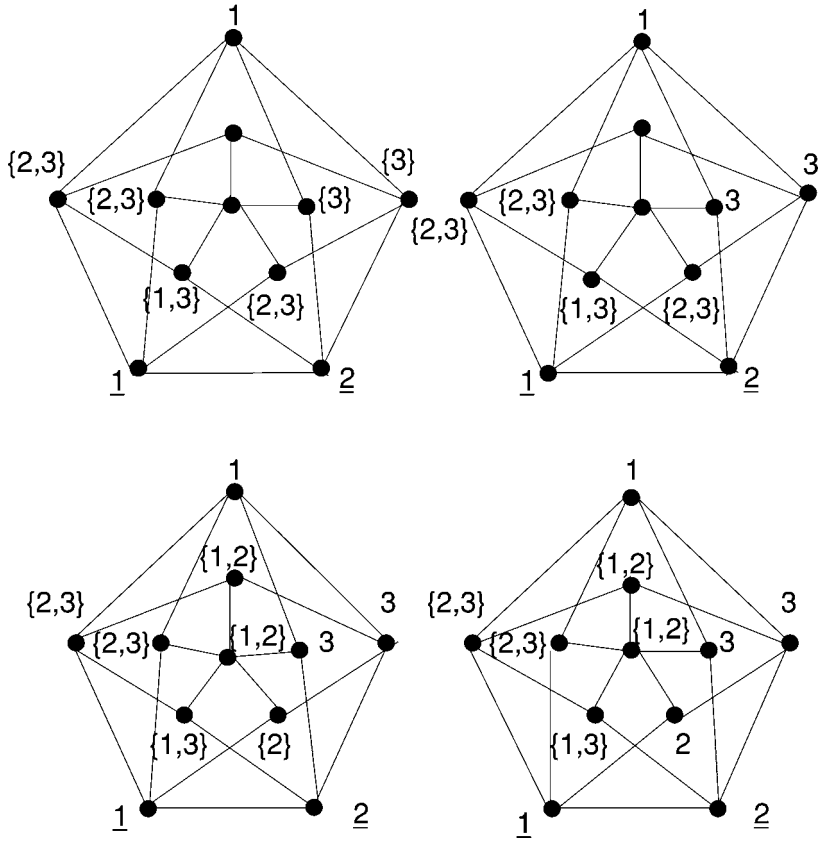

Figure 2. $\mathcal{A}''$ run on Myciel3.

Referring to the notation of Section 3, where we defined the starting CSP as $W' \cup N(W')$, we have that W' in this case is represented by W, and $N(W')$ by $N(W)$.

From figure 5 it can be noted that G_N is assigned a feasible coloring. Thus, $\mathcal{A}_k$ is invoked as described in Section 3. After the execution of $\mathcal{A}_k$, G_{N^2} is not edge consistent as all the nodes in $N^2(W)$ remain with empty domains. Hence, *AEC* is executed to compute a feasible solution for G_{N^2}, and the 4-coloring reported in figure 6 is obtained.

Now, we want to verify whether an improved lower bound can lead to a better (feasible) coloring. For this purpose we use $\mathcal{A}'$ to try to increase the clique bound. Figure 5 reports its execution which returns empty domains and allows the current lower bound $\omega(G)$ to be raised to 3.

It is easy to ascertain (see figure 7) that the largest CSP for which $\mathcal{A}'$ is not able to prove infeasibility is formed by $W \cup N(W)$. As can be simply verified, G_N is edge consistent and is a forest, thus it admits a backtrack free coloring (see figure 7).

At this point, $\mathcal{A}_k$ is invoked. The subgraph G_{N^2} is edge consistent and is a forest, and, as with G_N, it admits a feasible solution (see figure 8). Moreover, the 3-coloring found is an optimal solution for G, as the lower bound equals the upper bound.

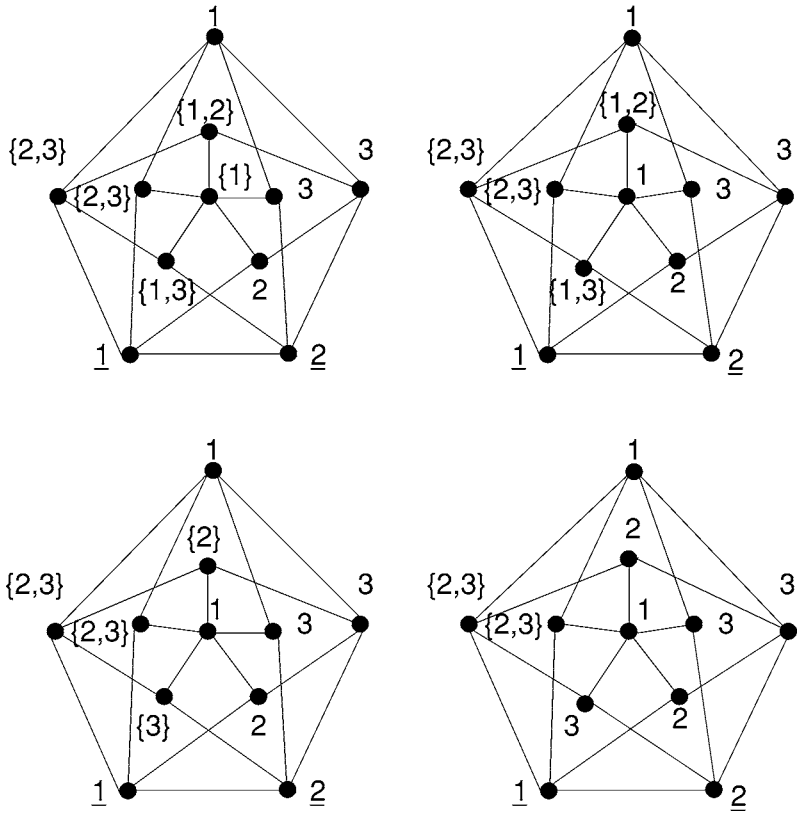

Figure 3. $\mathcal{A}''$ run on Myciel3.

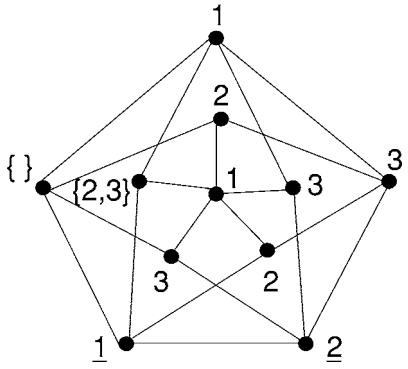

Figure 4. $\mathcal{A}''$ run on Myciel3.

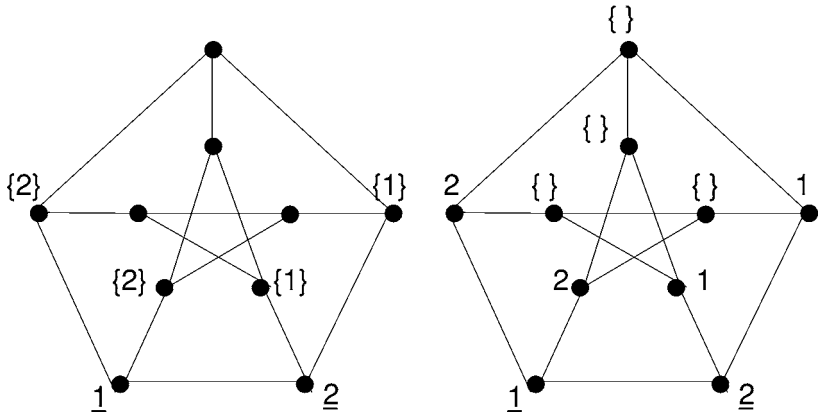

Figure 5. $\mathcal{F}$ run on Petersen graph.

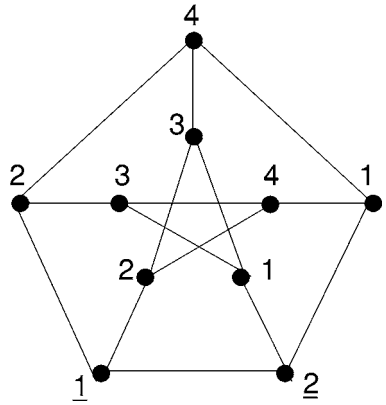

Figure 6. $\mathcal{F}$ run on Petersen graph.

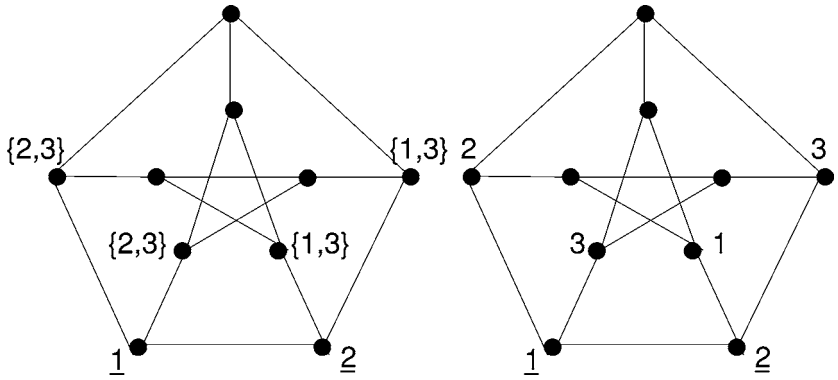

Figure 7. $\mathcal{F}$ run on Petersen graph.

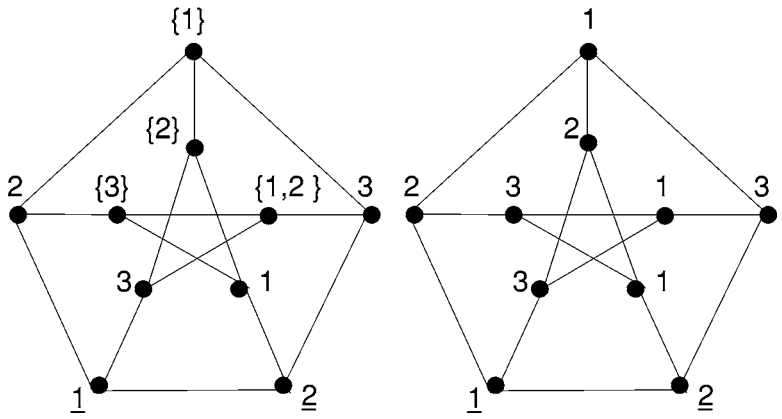

Figure 8. $\mathcal{F}$ run on Petersen graph.

Summarizing the results of this example, a 4-coloring is obtained from a lower bound equal to 2, while a 3-coloring is met from a lower bound equal to 3, thus confirming that $\mathcal{F}$ gives better upper bounds in correspondence to better lower bounds.

5. Conclusions

In the paper we have presented constraint propagation algorithms to compute new lower bounds on the chromatic number of a graph. Following the same constraint propagation technique, we have designed algorithms which find feasible colorings relying on backtrack free conditions for networks of binary constraints. We experimented with general random graphs, triangle free random graphs and benchmark graphs with $\chi(G) > \omega(G)$. Although easy to implement, it appears that the proposed algorithms are able to provide lower bounds which always dominate the clique bound.

Acknowledgement

The authors thank anonymous referees for their valuable suggestions that have led to this improved version.

References

Babel, L., (1991). "Finding Maximum Cliques in Arbitrary and in Special Graphs." *Computing* 46, 321–341.

Balas, E. and J. Xue. (1996). "Weighted and Unweighted Maximum Clique Algorithms with Upper Bounds from Fractional Coloring." *Algoritmica* 15(5), 397–412.

Baptiste, P., C. Le Pape, and W. Nuijten. (1995). "Incorporating Efficient Operations Research Algorithms in Constraint-Based Scheduling." In *Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research*, Timberline Lodge, Oregon.

Caramia, M. (2000). "Coloring Algorithms and Their Application to Project Scheduling." Ph.D. Thesis, University of Rome "La Sapienza".

Caramia, M. and P. Dell'Olmo. (1999). "A Fast and Simple Local Search for Graph Coloring." In *Proc. of the 3rd Workshop on Algorithm Engineering* (WAE'99), Lecture Notes in Computer Science, Vol. 1668, pp. 319–333.

Caramia, M. and P. Dell'Olmo. (2001a). "Iterative Coloring Extension of a Maximum Clique." *Naval Research Logistics* 48(6), 518–550.

Caramia, M. and P. Dell'Olmo. (2001b). "A Lower Bound on the Chromatic Number of Mycielski Graphs." *Discrete Mathematics* 235, 79–86.

Caramia, M., P. Dell'Olmo. and G.F. Italiano. (1999). "Tuning the Cache Performance of a Local Search Graph Coloring Algorithm." Technical Report No. 389, Centro Vito Volterra, University of Rome "Tor Vergata".

Caramia, M., P. Dell'Olmo. and G.F. Italiano. (2000). "CHECKCOL: Improved Local Search for Graph Coloring." submitted.

Cooper, M.C. (1997). "Fundamental Properties of Neighborhood Substitution in Constraint Satisfaction Problems." *Artificial Intelligence* 90, 1–24.

Dechter, R. and J. Pearl. (1988). "Network-Based Heuristics for Constraint-Satisfaction Problems." *Artificial Intelligence* 34, 1–38.

DIMACS ftp site for benchmark graphs, ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/.

DIMACS ftp site for benchmark machines, ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/volume/machine.

Fleurent, C. and J.A. Ferland. (1995). "Genetic and Hybrid Algorithms for Graph Coloring." *Annals of Operations Research* 63, 437–461.

Fleurent, C. and J.A. Ferland. (1996). "Object-Oriented Implementation of Heuristic Search Methods for Graph Coloring, Maximum Clique and Satisfiability." In D.S. Johnson and M.A. Trick (eds.), *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, AMS, Vol. 26, pp. 619–652.

Freuder, E.C. (1989). "A Sufficient Condition of the Backtrack Free Search." *J. ACM* 29, 24–32.

Gardner, M. (1969). *The Unexpected Hanging and Other Mathematical Diversions*. New York: Simon and Schuster.

Garey, M.R. and D.S. Johnson, (1979). *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-completeness.* San Francisco: W.H. Freeman & Company.

Garey, M.R. and D.S. Johnson, and L. Stockmeyer. (1976) "Some Simplified $\mathcal{NP}$-Complete Graph Problems." *Theor. Comput. Sci.* 1, 237–267.

Gondran, M. and M. Minoux. (1984). *Graphs and Algorithms*. New York: John Wiley and Sons.

Johnson, D.S, C.R. Aragon, L.A. McGeoch, and C. Schevon. (1991). "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning." *Operations Research* 39, 378–406.

Kratochvil, J. (1993). "Precoloring Extension with Fixed Color Bound." *Acta Math. Univ. Comenianae* 2, 139–153.

Mackworth, A.K. (1976). "Consistency in Networks of Relations." *Artificial Intelligence* 8, 99–118.

Mehrotra, A. and M.A. Trick. (1996). "A Column Generation Approach for Graph Coloring." *INFORMS J. on Computing* 8, 344–354.

Morgenstern, C. (1996). "Distributed Coloration Neighborhood Search." In D.S. Johnson and M.A. Trick (eds.), *DIMACS Series in Computer Mathematics and Theoretical Computer Science*, AMS, Vol. 26, pp. 335–358.

Mycielski, J. (1955). "Sur le Coloriage des Graphes." *Colloquim Mathematiques* 3, 161–162.

Sager T.J. and S. Lin. (1993). "A Pruning Procedure for Exact Graph Coloring." *ORSA J. Computing* 3, 226–230.

Sewell, E.C. (1996). "An Improved Algorithm for Exact Graph Coloring." In D.S. Johnson and M.A. Trick (eds.), *DIMACS Series in Computer Mathematics and Theoretical Computer Science*, AMS, Vol. 26, pp. 359–373.

Tutte, W.T. (1954). (B. Descartes) "Solution of Advances Problem No. 4526." *Advanced Mathematical Monthly* 61, p. 352.